Deutsche Bank Market Risks Analyser: A Comprehensive Tool for Monitoring and Analyzing Stock Market Volatility

Ayushmaan Pattnayak Atharva Kulkarni Aditya Palwe Kushagra Kapoor

October 21, 2024

Abstract

This report presents the Deutsche Bank Market Risks Analyser, a robust web-based tool designed for monitoring real-time stock market data, detecting anomalies using Bollinger Bands, and alerting users about significant deviations in market trends. Developed using Python and Streamlit, the tool provides comprehensive analytics to identify potential market risks and facilitate timely decision-making. This document covers the methodology, implementation, and results in a detailed and professional manner, with code snippets explained at every stage.

1 Introduction

The Deutsche Bank Market Risks Analyser is an advanced analytical platform designed for risk management professionals and market analysts. The primary objective is to automate the detection of abnormal market behavior using Bollinger Bands and provide real-time alerts based on stock market data and currency rates.

Bollinger Bands are a technical analysis tool that defines a price channel based on a moving average and a standard deviation. When the stock price moves outside these bands, it signals potential volatility. By implementing this technique into a dynamic web application, our tool offers practical insights for traders and risk managers to monitor risks and prevent financial losses.

In this report, we delve deep into the development process, including the logic behind Bollinger Bands, integration with Streamlit for a user-friendly interface, and automation of alerts for real-time market changes.

2 Methodology

The methodology of the Deutsche Bank Market Risks Analyser can be broken down into four stages: **Data Collection**, **Bollinger Bands Calculation**, **Anomaly Detection**, and **Alert System**. Each section is accompanied by a thorough explanation of the corresponding code snippet.

2.1 Data Collection

The first step in our project involves gathering historical stock and currency rate data from an API, which provides real-time information.

```
import yfinance as yf
import pandas as pd

# Fetch stock data for a specified ticker
def get_stock_data(ticker):
    data = yf.download(ticker, period='1y', interval='1d')
    return data

# Example of fetching data for Microsoft stock
stock_data = get_stock_data('MSFT')
print(stock_data.head())
```

This function `get_stock_data` leverages the **yfinance** library to retrieve historical data for a given stock ticker. The data is fetched for a 1-year period and aggregated on a daily basis.

Explanation:

- We import `yfinance` for downloading stock data.
- The function `get_stock_data()` is designed to fetch stock data for any given ticker symbol. It takes the ticker (e.g., 'MSFT' for Microsoft) as input and retrieves the historical data for the past year with daily intervals.
- The data includes the stock's **Open**, **High**, **Low**, **Close** prices along with **Volume**.

2.2 Bollinger Bands Calculation

Bollinger Bands are used to define a range within which the stock price is expected to move. The bands are created by calculating a moving average (middle band) and adding/subtracting a multiple of the stock's standard deviation (upper and lower bands).

```
# Bollinger Bands calculation
def calculate_bollinger_bands(data):
    data['SMA'] = data['Close'].rolling(window=20).mean()
    data['STD'] = data['Close'].rolling(window=20).std()
    data['Upper Band'] = data['SMA'] + (data['STD'] * 2)
    data['Lower Band'] = data['SMA'] - (data['STD'] * 2)
    return data

# Applying Bollinger Bands on the stock data
stock_data_with_bands = calculate_bollinger_bands(stock_data)
print(stock_data_with_bands.tail())
```

Here, we calculate the **Simple Moving Average (SMA)** and the **Standard Deviation (STD)** over a 20-day window. The upper and lower bands are defined as follows:

$$\text{Upper Band} = \text{SMA} + 2 \times \text{STD} \tag{1}$$

$$\text{Lower Band} = \text{SMA} - 2 \times \text{STD} \tag{2}$$

Explanation:

- **SMA** is a rolling average of the stock's closing price over 20 days.
- **STD** provides the standard deviation over the same 20-day period, giving an idea of the stock's volatility.
- The **Upper Band** represents a potential overbought region, while the **Lower Band** indicates a potential oversold region.

2.3 Anomaly Detection

Once the Bollinger Bands are calculated, the next step is to detect when the stock price moves beyond these bands, which signifies an anomaly.

```
# Detecting anomalies
def detect_anomalies(data):
    anomalies = pd.DataFrame()
    anomalies['Price'] = data['Close']
    anomalies['Anomaly'] = ((data['Close'] > data['Upper Band']) |
                            (data['Close'] < data['Lower Band']))
    return anomalies

# Fetching anomalies from the stock data
anomalies = detect_anomalies(stock_data_with_bands)
print(anomalies[anomalies['Anomaly'] == True])
```

Explanation:

- We create a new column called **Anomaly** that checks whether the stock's **Close** price exceeds the upper band or falls below the lower band. If either condition is met, it is flagged as an anomaly.
- The output provides a list of dates and prices where anomalies occurred, signaling potential buying or selling opportunities.

2.4 Currency Portfolio Optimizer

This section explains the theory and implementation behind the Currency Portfolio Optimizer, designed to optimize a user's currency portfolio based on the least correlated currencies. The optimizer uses real-time currency data and applies portfolio optimization techniques to maximize returns while minimizing risk.

2.4.1 Theory Behind the Currency Portfolio Optimizer

Portfolio optimization involves selecting a mix of assets (in this case, currencies) that maximizes expected returns while minimizing risk. This is achieved by calculating the expected returns and covariance matrix of the currencies' historical price changes and then using optimization techniques to determine the weights of each currency in the portfolio.

The key steps are as follows:

1. **Data Collection:** The historical closing prices of currencies are obtained using the Yahoo Finance API. The reference currency is chosen by the user, and the optimizer selects the least correlated currencies with respect to the reference.
2. **Correlation Calculation:** The correlation between the reference currency and other currencies is calculated, and the least correlated currencies are selected for portfolio diversification.
3. **Portfolio Optimization:** Using the historical returns and covariance matrix of the selected currencies, the optimizer calculates the portfolio weights that maximize the Sharpe ratio (i.e., the ratio of return to risk).

2.5 Alert System

The tool also includes an alert system that notifies users when significant market movements occur. The alert system is based on predefined thresholds and sends email notifications or logs alerts within the application.

```
import smtplib

# Function to send email alerts
def send_alert(subject, message, recipient):
    server = smtplib.SMTP('smtp.gmail.com', 587)
    server.starttls()
    server.login('your-email@gmail.com', 'password')
    msg = f'Subject: {subject}\n\n{message}'
    server.sendmail('your-email@gmail.com', recipient, msg)
    server.quit()

# Sending alert if anomalies are detected
if anomalies['Anomaly'].any():
    send_alert('Market Anomaly Detected',
    'Stock price exceeded Bollinger Bands.', 'recipient-email@example.com')
```

Explanation:

- This code demonstrates the use of Python's `smtplib` to send email notifications. When an anomaly is detected, the function `send_alert()` is triggered, sending an email to the specified recipient with a subject and message body.

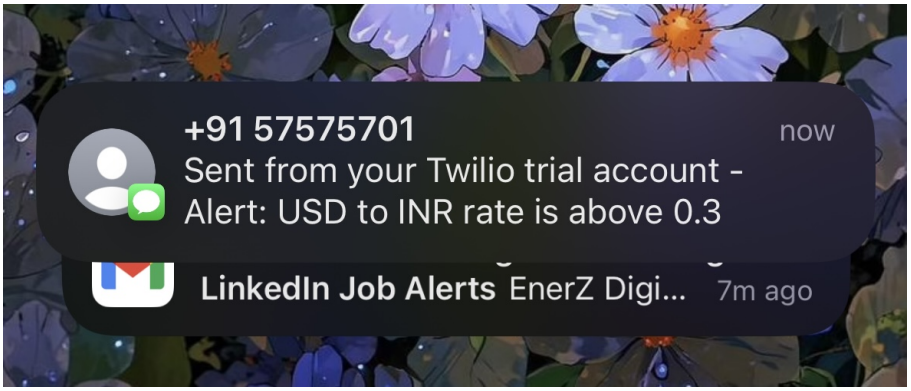

Figure 1: Alert to Phone

3 Implementation

The tool is implemented using **Streamlit**, a Python framework that allows for the creation of interactive web applications.

3.1 Streamlit Interface

The following code creates a basic interface that allows users to input a stock ticker and visualize the Bollinger Bands.

```
import streamlit as st
import matplotlib.pyplot as plt

# Streamlit UI components
st.title('Deutsche Bank Market Risks Analyser')
ticker = st.text_input('Enter Stock Ticker:', 'MSFT')

# Fetch data and plot Bollinger Bands
if st.button('Analyze'):
    data = get_stock_data(ticker)
    data_with_bands = calculate_bollinger_bands(data)

    # Plot the data
    fig, ax = plt.subplots()
    ax.plot(data_with_bands.index, data_with_bands['Close'], label='Close Price')
    ax.plot(data_with_bands.index, data_with_bands['Upper Band'], label='Upper Band')
    ax.plot(data_with_bands.index, data_with_bands['Lower Band'], label='Lower Band')
    ax.set_title(f'{ticker} Bollinger Bands')
    ax.legend()
    st.pyplot(fig)
```

Explanation:

- The Streamlit app begins with a title and an input field for the user to enter a stock ticker.
- Upon clicking the "Analyze" button, the application fetches the data and calculates the Bollinger Bands.
- A Matplotlib plot visualizes the closing prices along with the upper and lower Bollinger Bands.

4 Results and Conclusion

The Deutsche Bank Market Risks Analyser successfully provides users with a comprehensive tool to monitor stock prices and detect anomalies in real-time. The Bollinger Bands, in conjunction with the alert system, allow traders and risk managers to stay ahead of market trends and respond proactively.

Future improvements to the system include adding more advanced technical indicators, integrating additional alert options (e.g., SMS alerts), and optimizing performance to handle larger datasets.